On Topology Reconfiguration for Defect-Tolerant NoC-Based Homogeneous Manycore Systems

Lei Zhang, Yinhe Han, *Member, IEEE*, Qiang Xu, *Member, IEEE*, Xiao wei Li, *Senior Member, IEEE*, and Huawei Li, *Member, IEEE*

***Abstract*—Homogeneous manycore systems are emerging for tera-scale computation and typically utilize Network-on-Chip (NoC) as the communication scheme between embedded cores. Effective defect tolerance techniques are essential to improve the yield of such complex integrated circuits. We propose to achieve fault tolerance by employing redundancy at the core-level instead of at the microarchitecture level. When faulty cores exist on-chip in this architecture, however, the physical topologies of various manufactured chips can be significantly different. How to reconfigure the system with the most effective NoC topology is a relevant research problem. In this paper, we first show that this problem is an instance of a well known NP-complete problem. We then present novel solutions for the above problem, which not only maximize the performance of the on-chip communication scheme, but also provide a unified topology to Operating System and application software running on the processor. Experimental results show the effectiveness of the proposed techniques.**

***Index Terms*—Defect tolerance, manycore system, network-on-chip, core-level redundancy, topology reconfiguration.**

I. Introduction

As TECHNOLOGY advances, industry has started to employ multiple cores on a single silicon die in order to improve performance through parallel execution, which has the benefits of power-efficiency and short time-to-market [1]. Significant research has been undertaken on tera-scale computing that is able to integrate tens to hundreds of homogeneous processing cores on a single chip to process massive amounts of information in parallel [2], [3]. For example, an 80-core teraflop processor prototype was demonstrated at Intel Developer Forum 2006 [4]. Such processors containing a large number of cores are called manycore processors (note the difference from multicore processors that contain a small number of cores). In terms of communication infrastructure, Network-on-Chip (NoC) is generally regarded as the most promising interconnect solution for giga-scale Integrated Circuits (ICs) such as manycore processors [5], [49], in which the topology determines the ideal performance of the on-chip network whereas the routing algorithm and the flow control mechanism determine how much of this potential is realized. As a result, Operating System (OS) should understand the topology of NoC-based manycore systems to dispatch and schedule tasks to multiple cores more effectively; while programmers should also be aware of the topology to improve the performance of parallel applications [16], [52].

There are many challenges for the architecture design of these NoC-based manycore systems, in which fabrication yield is one of the most serious concerns because an IC's profitability depends heavily on it [6], [7]. With the ever-increasing circuit density, obtaining high fabrication yield solely through improving the manufacturing process is increasingly difficult and will become unaffordable in the near future. For example, as stated in [8], it would have been lucky to get yield in the range of 10–20 percent for the Cell processor if architectural help is not provided. A more practical solution is therefore to provide defect tolerance capabilities on-chip by incorporating redundant circuits. For example, Memory Built-In-Self-Repair (MBISR) techniques have been widely utilized in the industry and proved to be very effective to keep the high fabrication yield of memory circuits. Such techniques should be extended to other types of VLSI circuits as well [9].

However, tolerating defects in the microprocessor is quite different from tolerating defects in memory because the processor's internal structure is not as regular as memory cells, and previous attempts in this domain mainly focused on introducing microarchitecture-level redundancy (e.g., [10], [11]). This is appropriate for multicore chips (e.g., a quad-core processor) in order to keep the overhead small. When the number of on-chip cores increases to a point that single core becomes inexpensive when compared to the entire chip (e.g., a 64-core processor), however, it is not necessary to tolerate defective cores at the microarchitecture level. Instead, it is more appropriate to employ core-level redundancy in such case to reduce the complexity associated with microarchitecture-level redundancy.

Manuscript received December 10, 2007; revised April 07, 2008. First published June 16, 2009; current version published August 19, 2009. This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant (60633060, 60606008, 60776031, 60803031, 60806014, and 60831160526), in part by National Basic Research Program of China (973) under Grant 2005CB321604, and in part by Hi-Tech Research and Development Program of China (863) under Grant (2007AA01Z109, 2007AA01Z107, 2007AA01Z476, 2007AA01Z113, and 2009AA01Z126). The work of Y. Han was also supported by the fund of Chinese Academy of Sciences due to the President Scholarship. The work of Q. Xu was supported in part by the General Research Fund CUHK417406, CUHK417807, and CUHK418708 from Hong Kong SAR Research Grants Council (RGC), in part by NSFC under Grant 60876029, and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme. A preliminary version of this paper was published in the Proceedings of the IEEE/ACM Design, Automation, and Test in Europe (DATE) Conference, March 2008.

L. Zhang, Y. Han, X. Li, and H. Li are with the Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (e-mail: zlei@ict.ac.cn; yinhes@ict.ac.cn; lxw@ict.ac.cn; lihuawei@ict.ac.cn).

Q. Xu is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. He is also with the CAS-CUHK Shenzhen Institute of Advanced Integration Technology, Shenzhen 518067, China (e-mail: qxu@cse.cuhk.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2008.2002108

For NoC-based manycore systems with core-level redundancy, faulty cores are replaced by spare ones placed on-chip. Therefore, it is possible that the topology of the target design is modified and different fabricated chips may have different underlying topologies. This is a big burden for programmers because an optimized program for one topology may not work well for a different one and the programmers are facing various topologies when optimizing their parallel programs.

To address the above problem, the concept of virtual topology is reintroduced from prior network embedding problem in this paper. A virtual topology is isomorphic with the topology of the target design but is a degraded version. From the viewpoint of OS and programmers, they always see a unified virtual topology regardless of the various underlying physical topologies. This eases the dispatching and scheduling tasks for OS and facilitates the optimization of parallel programs. The above issue was briefly discussed in [12]. When compared to [12], in this paper we re-define the problem by introducing two new metrics, namely Distance Factor (DF) and Congestion Factor (CF), to evaluate the performance of different virtual topologies. We also introduce new algorithms to tackle the problem, and conduct extensive simulation experiments to verify the effectiveness of the proposed solution.

The rest of this paper is organized as follows. Section II motivates our research work. In Section III, we formulate the topology reconfiguration problem investigated in this paper. Section IV reviews prior related work. Section V gives in-depth analysis of the formulated problem, which is shown to be an instance of a well-known NP-complete problem. The proposed topology reconfiguration algorithm is then described in detail in Section VI. Next, Section VII presents experimental results. Finally, we conclude this paper in Section VIII.

II. Motivation

A. Core-Level Redundancy in Homogeneous Manycore Processors

As the internal structure is not as regular as memory cells, previous research work on defect tolerance in microprocessors mainly focused on introducing microarchitecture-level redundancy. Redundancy improves yield while at the same time may reduce the chip performance. Researchers thus evaluate the effectiveness of various redundancy mechanisms using performance averaged yield (Y_{PAV}) [10] or Yield-Adjusted Throughput (YAT) [11]. Performance degradation is measured by the relative Instructions Per Cycle (IPC), i.e., the ratio of the reduced IPC to the maximum IPC of the perfect version.

For multicore and manycore processors, the chips themselves naturally have regularity and redundancy as they contain a number of cores. As a result, core-level redundancy could be employed besides microarchitecture-level redundancy. Microarchitecture- and core-level redundancy are named intra- and inter-processor redundancy respectively in [10]. In the former case, a core can be in any degraded states, but the entire chip is considered bad once the available intra-processor redundancy is exhausted in even one of its cores. In the latter case, a core becomes useless if it contains any faults. However, as long as enough of the remaining cores are functional, the chip is considered to be operational.

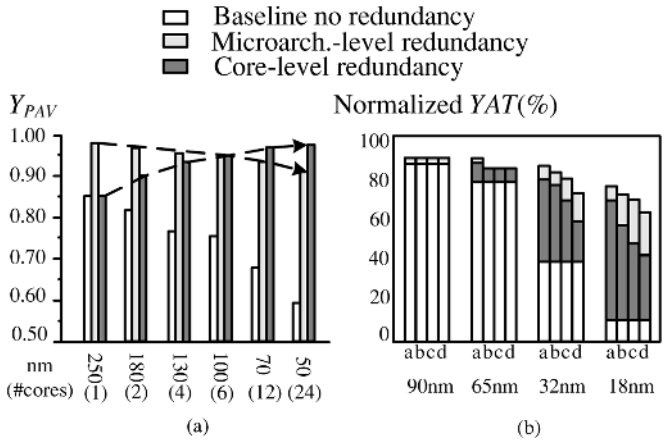


Fig. 1. Comparison between microarchitecture- and core-level redundancy. (a) Y_{PAV} comparison redrawn from [10] with the permission of the author. (b) YAT comparison redrawn from [11] with the permission of the author.

Various types of microarchitecture-level redundancies are considered with core-level redundancy by using poisson yield model in [10]. SPEC2000 and a speech recognition benchmark are chosen to get the IPC reduction. The results are reproduced and shown in Fig. 1(a). The x-axis shows the feature size and the number of cores per chip at each technology. As can be seen in the figure, although there are significant benefits by using microarchitecture-level redundancy when compared to baseline model, Y_{PAV} drops from 98% at 250 nm to 91.3% at 50 nm. Core-level redundancy covers the entire area of the chip and therefore Y_{PAV} increases uniformly from 85.4% to 98%. The yield benefits offered by microarchitecture-level and core-level redundancy crossover at 100 nm.

The authors in [11] proposed a novel defect tolerant microarchitecture (namely *Rescue*). Core-level redundancy (called "core sparing" in their work), is used to compare with *Rescue* by using HotSpot model and negative binomial yield model. IPC reduction is evaluated by simulating 23 benchmark programs from SPEC2000. It also assumes a 20%(a), 30%(b), 40%(c), and 50%(d) growth of core complexity starting from one core per chip at the 90 nm. The results are redrawn and shown in Fig. 1(b). Similarly, we can observe, as technology advances, YAT becomes increasingly lower without redundancy. At the same time, microarchitecture-level redundancy brings YAT improvement, but at a smaller scale when compared to core-level redundancy in newer technology generation. Microarchitecture-level redundancy shows greater improvement under larger core complexity growth, because the chip has fewer cores and each defective core disables a larger portion of the chip.

From the above analysis, we can conclude that, for manycore chips, because the number of on-chip cores is large and they are fabricated in latest technology, the probability of an embedded core being defective is quite small. Each degraded chip contains a majority of fully functional cores and a small number of defective ones. Therefor, it is not necessary to tolerate defective cores at the microarchitecture-level. Instead, it is more appropriate to

employ core-level redundancy in such case to reduce the complexity associated with microarchitecture-level redundancy.

In fact, industry has started to employ core-level redundancy in their products recently. For example, while the Cell processor contains eight Synergistic Processing Elements (SPEs), Sony's PlayStation 3 video game console considers using only seven of them to increase the manufacturing yield [8]. This approach is also applied in Sun's UltraSPARC T1 processor [13], [14] and Azul's Vega2 chip [15].

There are two schemes to design homogeneous multicore or manycore chips with core-level redundancy, namely *As Many As Available* (AMAA) and *As Many As Demand* (AMAD). The AMAA scheme, adopted in the T1 processor, degrades a chip by disabling faulty cores only. For example, a fabricated quad-core processor can be a full version with 4 functional cores; or it can be degraded to a tri-core, dual-core or single-core processor depending on the number of faulty cores. In AMAD scheme, also denoted as "$N + M$" mechanism in this paper, adopted in the Cell processor ($N = 7, M = 1$), an N-core processor is provided with M redundant cores and we always provide customers with N operational cores. That is, it is possible that there are fault-free cores left unused in AMAD.

It is preferred to employ the AMAA scheme in multicore to keep the overhead small. However, as the number of on-chip cores increases, the overhead of leaving a few redundant cores on-chip unused is acceptable because a single core is inexpensive compared to the entire chip as discussed above. In addition, with many cores implemented on-chip, we may get various types of degraded chips (with different number of faulty cores) after fabrication and the yield of the demanded N-core processor cannot be promised in AMAA scheme. Finally, from a commercial point of view, it may cause some confusion in marketing with many different degraded versions. Therefore, for manycore processors, AMAD scheme is preferred and we mainly focus on this scheme in this paper.

Manycore processors typically use NoC as the communication infrastructure, in which the topology determines the ideal performance whereas the routing algorithm and the flow control mechanism determine how much of this potential is realized. However, in AMAD scheme, as the cores that are fabricated to be defective are not known a priori, when they are replaced by spare cores, the topology of the target design can be different. For example, suppose we want to provide 9-core processors with 3×3 2D mesh topology to customers, as shown in Fig. 2(a). Also, suppose 3 redundant cores (1 column) are provided to improve the yield of these chips as shown in Fig. 2(b). If some cores (no more than 3) are defective, we could still get 9-core processors. However, as shown in Fig. 2(c), if faulty cores are replaced by spare cores, not only the topologies that we get are different from what we expect, but also the topologies of different chips can be distinct. These changed topologies become irregular and would cause performance degradation for manycore processors.

B. Topology Impacts on NoC-Based Manycore Systems

In NoC-based homogeneous manycore systems, the performance of the on-chip communication significantly affects the efficiency of parallel applications. As a result, to minimize the communication overhead among threads or tasks, today's OS relies on explicit knowledge of the underlying topology [52]. For example in Microsoft Windows Server 2003, a so-called Advanced Configuration and Power Interface (ACPI) circuit is used to pass a description of the physical topology of the system to OS [16]. The topology information is stored in Static Resource Affinity Table (SRAT), and is used by Windows when dispatching and scheduling tasks. For example, a representative scheme, namely Gang Scheduling [17], divides processors into groups, in which processors of the same group have lower communication overhead. Tasks that frequently communicate with each other will be assigned to processors in the same group to minimize communication overhead.

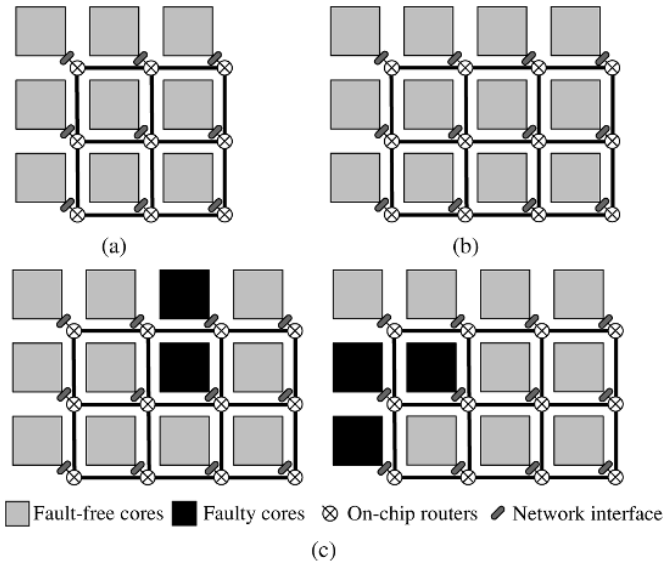


Fig. 2. Faulty cores change the topology of target design. (a) What we expect. (b) What we implement. (c) What we get.

In addition, from the parallel programmers' perspective, to optimize the performance of the application software, currently they need to know the underlying manycore's organization [51]. For example, topology information is provided to programmers through API functions in Windows Server 2003. This is the communication-exposed programming for NoC platforms [49]. Such tailored programs may be not portable to other processors due to different system architectures, such as the number of on-chip cores and their topology.

C. Physical Topology and Virtual Topology

As shown in Fig. 2, faulty cores change the target topology and different chips may have distinct underlying topologies. It would be rather cumbersome for OS and programmers to face various different topologies and optimize them differently. To address this problem, we propose to provide a unified virtual topology regardless of the underlying one. Before introducing the details, we first define *Reference Topology* as the topology of the target design that we expect. For example, the 3×3 2D mesh topology in Fig. 2(a) is the expected reference topology.

For the illustrative "$9 + 3$" manycore processor shown in Fig. 2(b), suppose the 7th, 10th and 11th cores are defective

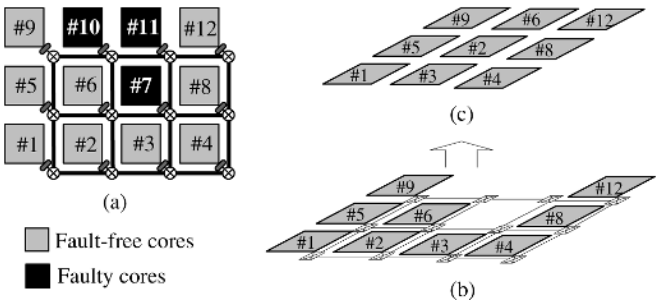


Fig. 3. Physical topology and virtual topology. (a) A chip with faulty cores. (b) The physical topology. (c) A virtual topology.

after fabrication as shown in Fig. 3(a), these cores are considered to be removed out of the chip. The remaining fault-free cores and their interconnections construct a *Physical Topology* as shown in Fig. 3(b). It should be emphasized that once a manycore processor is taped out, its physical topology is determined and cannot be changed during its lifetime. This is fundamentally different from board-level multiprocessor systems, which are much easier to be repaired since the target topology can be maintained by simply replacing the faulty processor with a good one.

Based on our AMAD scheme, a 9-core processor can still be provided but with different topology when compared to the reference topology. That is, we can construct a *Virtual Topology* of the chip based on the given physical topology, which is isomorphic with the reference topology. An example is shown in Fig. 3(c), in which we construct a *virtual* 3×3 2D mesh topology.

With the above configuration, the 3rd, the 5th, the 6th and the 8th cores are four virtual neighbors of the 2nd core. The 3rd core is considered to be below the 2nd core virtually, although it locates at the 2nd core's right-hand side physically. In addition, while the 5th core is more than one hop away from the 2nd core, they are considered to be adjacent in the above virtual topology.

By using virtual topology, OS and programmers always see a unified topology that is isomorphic with the reference topology, no matter how the underlying cores are connected physically. This greatly simplifies task dispatching and scheduling duties for OS and also facilitates the optimization of parallel programs. In addition, a unified topology that isolates various physical topologies for different chips also significantly eases marketing process.

A similar idea has been applied in Cray T3E network [18]. If some processors fail during the operation of the system, one or more of them may not be physically contiguous. To continue providing applications with a contiguous range of virtual processor numbers, the routing table along with the logical "who am I" registers allows the nodes to be logically renamed, i.e., mapping from physical to virtual number. This kind of "hot swapping" is totally transparent to users. As mentioned above, the failure of nodes and the change of topologies in systems such as Cray T3E are temporary and can be easily recovered because a faulty processor is removed from the system and replaced while OS and user jobs are kept running on the healthy nodes. However, for manycore processors, defects are permanent and physical topologies cannot be recovered.

It should be also noted that, depending on the architecture design of manycore processors, there are many ways to implement the mapping from various physical topologies to their corresponding virtual topology. For example, one possible solution is to add a firmware layer below OS to record mapping information which is obtained after fabrication test. This is similar to the CORE_AVAILABLE_REG used in UltraSPARC T1 processor [13], [14]. OS and programmers always work on the reference topology while the firmware is responsible for transformation.

III. Problem Formulation

On-chip faulty cores change the topology of the target design and cause performance degradation for parallel applications. To tackle this problem, we use virtual topology to provide a unified interface to OS and programmers, no matter how the underlying cores are connected physically. At the same time, however, as there can be many virtual topologies for a particular physical topology and they may affect applications differently, we should choose the one that results in the best performance.

Since there are a wide range of applications with different characteristics running on the NoC-based manycore systems and they may have different requirements on the construction of virtual topologies, it is difficult to evaluate the impact of virtual topologies on various applications at the chip architecture design stage. As a result, we evaluate the performance of virtual topologies themselves and mainly consider the average latency and throughput of different virtual topologies.

In order to do so, from the viewpoint of the NoC, two evaluation metrics are introduced in this section to model the performance degradation of different virtual topologies when compared to the reference topology, namely Distance Factor (DF) and Congestion Factor (CF). For the sake of simplicity, we assume the communication infrastructure to be fault-free in this research work. This assumption can be justified since the routers and links use much less hardware resources when compared to the cores and are thus less vulnerable to defects [32]. Also, it would not cause significant overhead to include fault-tolerant features such as Triple Modular Redundancy (TMR) to protect them.

Distance Factor: The zero-load latency T_0 of a topology can be expressed as [48]: $T_0 = H \times t_r + D/v + L/b$. It is composed of three terms. The router delay is $H \times t_r$ for a network with an average hop count of H and a delay of t_r through a single router. The time of flight is D/v for a network with an average distance of D and a propagation velocity of v. The last one is the serialization latency which is the time for a packet of length L to cross a channel with bandwidth b.

For a particular physical topology, virtual topologies differ from each other only in the average hop count H. When compared to reference topology, it is obvious that the average hop count of an irregular virtual topology becomes larger and thus the zero-load latency becomes longer. The distance factor is used to evaluate such degradation, in which $\mathrm{DF}_{nn'}$ between two nodes n and n' is defined as the physical hops between them

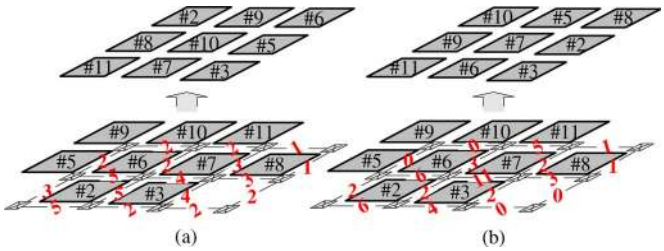


Fig. 4. CF comparison between two virtual topologies with the same DF $(\mathrm{DF} = 2)$ for a given physical topology. (a) Virtual topology I. (b) Virtual topology II.

$(\mathrm{DF}_{nn'} = Hops_{nn'})$ and the distance factor of node n (DF_n) is defined as the average distance factor between node n and all its k virtual neighbors

$$\mathrm{DF}_n = \frac{1}{k}\sum_{n'=1}^{k} \mathrm{DF}_{nn'}. \tag{1}$$

Finally, the distance factor of a virtual topology (DF) is defined as the average DF_n of all nodes

$$\mathrm{DF} = \frac{1}{N}\sum_{n=1}^{N} \mathrm{DF}_n. \tag{2}$$

(There are in total *N* nodes in the virtual topology.)

The reference topology has the minimum DF as usually virtual neighbors are located next to each other physically. For example, DF is 1 in mesh and torus topologies, which means that each pair of virtual neighbors is exactly one hop away from each other. Larger value of DF means longer communication delay among virtual neighbors.

Congestion Factor: For a given physical topology, it is likely that there are several virtual topologies with the same DF values, as shown in Fig. 4. We therefore use congestion factor to further evaluate the performance of virtual topologies. A virtual topology not only changes the average hop count among cores but also affects the distribution of channel load. Traffic may become unbalanced among different links. As the more balanced the channel load, the closer the throughput of the network is to the ideal case [48], a virtual topology that could balance traffic more evenly across all NoC links is preferred.

According to the previous discussion, traffic distribution in NoC-based manycore systems has the property of spatial locality, i.e., communication is more likely to happen between adjacent cores rather than distant ones. We thus only consider the case where a node only communicate with its virtual neighbors. We define the congestion factor of a physical link l (denoted as CF_l) as follows: for any nodes n and n', if they are virtual neighbors, and l is on one of the routing paths between them according to the NoC's routing mechanism (e.g., XY-routing [49]), we add CF_l by 1. For the two virtual topologies in Fig. 4, the CF_l values are shown above each physical links. It is clear that traffic in topology I is much balanced than the one in topology II. In topology II, some links are much congested $(\mathrm{CF}_l = 11)$ while some others are barely used $(\mathrm{CF}_l = 0)$.

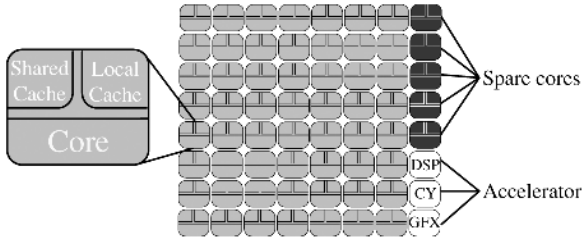


Fig. 5. System organization for manycore platform with "$N + M$" scheme.

Based on the above observation, we define the congestion factor of a virtual topology (CF) as the standard deviation of CF_l of all links to indicate the traffic distribution

$$\mathrm{CF} = \sqrt{\frac{\sum_{l=1}^{L}(\mathrm{CF}_l - \overline{\mathrm{CF}_l})^2}{L-1}}. \tag{3}$$

(There are in total L links in the physical topology.)

CF of the reference topology is 0, which means that traffic can be more balanced across the network[1]. Greater CF means less even flow distribution. Please note that even though advanced routing algorithms can be introduced to balance channel load, CF can be an auxiliary performance metric to evaluate the raw flow distribution which reflects the quality of a virtual topology.

With the above two metrics, the quality of different virtual topologies can be evaluated and compared. DF and CF might be conflicted with each other during optimization, hence we unify them together. The Unified Metric (UM) is defined as

$$\mathrm{UM} = w_{\mathrm{DF}} \times \mathrm{DF} + w_{\mathrm{CF}} \times \mathrm{CF} \tag{4}$$

in which w_{DF} and w_{CF} are the optimization weights designated by users $(w_{\mathrm{DF}} + w_{\mathrm{CF}} = 1)$.

Reconfiguration from physical to virtual topology is very complex and it depends heavily on the system organizations, such as the reference topology, the on-chip redundancy distribution, etc. In this paper, we mainly focus on mesh and torus topologies, which are the most widely used ones in NoC-based manycore systems. We adopt a representative scalable manycore architecture proposed by Intel as our platform model, which integrates an array of tens to hundreds of streamlined processing cores and accelerators connected by a scalable NoC infrastructure [4], as shown in Fig. 5. We formulate the topology reconfiguration problem for 2D mesh/torus topology investigated in this paper as follows:

[Topology Reconfiguration Problem (TRP)]: For an $R \times C$ homogeneous manycore processor with S redundant cores, suppose D cores $(D \le S)$ are faulty, construct $R \times C$ coordinates as follows:

$$\begin{bmatrix} (R-1,0) & (R-1,1) & \cdots & (R-1,C-1) \\ \cdots & \cdots & \cdots & \cdots \\ (1,\,0) & (1,\,1) & \cdots & (1,\,C-1) \\ (0,\,0) & (0,\,1) & \cdots & (0,\,C-1) \end{bmatrix}.$$

[1]Please note, congested links are usually revealed around the middle of network even for uniform traffic pattern in practice, CF metric is mainly for comparison purpose and 0 is its ideal upper bound.

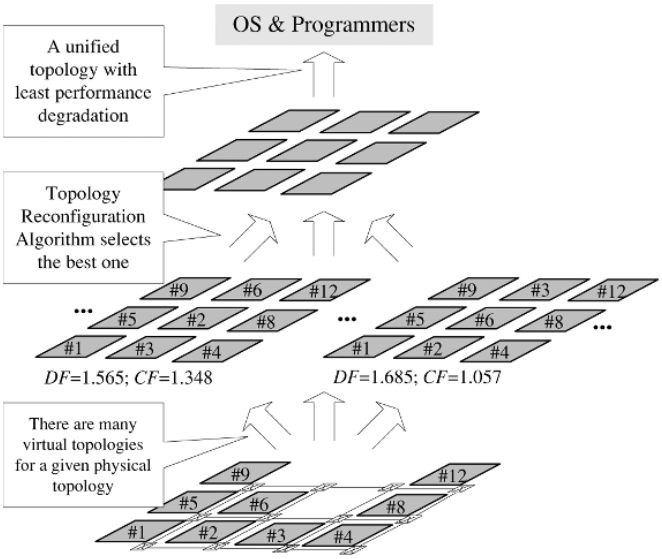


Fig. 6. Topology reconfiguration.

Distribute these coordinates to $(R \times C + S - D)$ fault-free cores to construct a virtual topology T_{virtual}, in which nodes with coordinates $(i+1, j), (i-1, j), (i, j+1)$ and $(i, j-1)$ are four virtual neighbors of node (i, j), and nodes without being assigned coordinates are left unused, satisfying

$$\text{UM of } T_{\text{virtual}} \text{ is minimized.}$$

Two example virtual topologies for a given physical topology are shown in Fig. 6. The values of DF and CF for these two virtual topologies are also shown in the figure. Clearly, new topology reconfiguration algorithm needs to be developed to select the best candidate topology. Before introducing our proposed algorithms, we firstly review prior related work in this area and then give some in-depth analysis of the above TRP problem in the following two sections.

IV. Related Prior Work

In this section, we briefly review related prior research work, including the defect tolerance for memory chips and VLSI array processors and the network embedding problems. We show the similarities and differences between the topology reconfiguration problem studied in this paper with previous work.

A. Defect Tolerance in Memory and VLSI Array Processors

Using redundant components to achieve yield improvement has been widely applied in memory chips and VLSI array processors for a long time.

To avoid yield loss in memory chips, spare elements, i.e., redundant columns, rows, words or small blocks are added to repair faulty storage cells for almost all memories with relatively high capacity [50]. In MBISR, failure bitmap information obtained through test is stored on-chip for repair purpose. The repair efficiency is determined by spare structure, redundancy analysis and repair strategy. 2D redundancy is the most widely used spare structure nowadays, in which both spare rows and spare columns are employed. The objective of redundancy analysis is to choose the minimum number of spare rows and columns that cover all the faulty cells. The complexity of 2D redundancy analysis problem has been proved to be NP-complete [19]. Finally, the time required to determine the repair solution is also a crucial factor. Lots of research work has been dedicated to the above areas [20]–[24].

A VLSI array processor integrates a large number of simple Processing Elements (PE) on a single chip or silicon wafer. To improve its yield, redundant PEs are often provided and fabrication-time reconfiguration techniques are applied to repair faulty PEs with spare ones [25]. There are generally two approaches to reconfigure VLSI array processors, namely the redundancy approach and the degradation approach. In redundancy approach, some PEs are dedicated as spare parts, and if these PEs cannot replace all the faulty ones, the chip has to be discarded. Various reconfiguration algorithms have been proposed in [26]–[29]. In the degradation approach, all PEs are treated in a uniform manner to derive a fault-free subarray, whose size is flexible. Two metrics, *harvest* and *degradation* are commonly used to evaluate the efficiency of reconfiguration algorithms in the degradation approach [30]–[32]. The *harvest* represents how effective the fault-free PEs are utilized to construct a subarray and the *degradation* measures the performance loss due to a smaller fault-free subarray than the original array.

For memory chips and VLSI array processors, their physical topologies have to be maintained the same before and after reconfiguration. The regularity of physical structures is required by the usages of such chips. The above reconfiguration problems differ significantly from the one for homogeneous manycore processors. This is because, every core in manycore processors is an autonomous system and is able to communicate with other cores through on-chip interconnection network. The physical topology is therefore not necessary to be kept the same after reconfiguration. Only a unified virtual topology should be maintained as described before.

B. Network Embedding Problems

The basic idea of constructing a virtual topology based on a physical topology for a certain purpose has been widely applied in many research areas. A famous application is the overlay networks [33], which create a structured virtual topology above the basic transport protocol level to facilitate deterministic content search. Virtual neighbor nodes in overlay networks are defined by identifiers derived from the stored contents. In this subsection, we briefly review the network embedding research problems that are closely related to our topology reconfiguration problem for manycore processors.

The network embedding problem, which has been studied extensively, is widely used for simulations between networks with different topologies. By embedding a G(uest) network topology into a H(ost) topology, parallel programs could have better portability. This is because one can automatically transform any parallel algorithms developed for the multiprocessor system with topology G into an algorithm for the system with topology H. [34] focused on embedding of any arbitrary network into its optimum complete binary trees. [39] proposed a new approach to embed a given torus into another given torus. [35] studied the embedding of rings and 2D mesh into a RP(k) network.

An application of network embedding in parallel computing is the mapping from virtual process topology to physical processor topology. The virtual process topology is the abstract of communications among processes or tasks, in which each vertex represents process, and an edge represents the communication between two processes. To execute a parallel program, its process topology should be constructed effectively based on the underlying processor topology. The virtual process topology is also supported by MPI libraries [36], [37] discussed the mapping problem in switch-based cluster systems with irregular topology. Ref. [38] presented techniques to reconfigure application topology in an octagonal 2D mesh machine topology when faults occur.

The topology reconfiguration problem studied in this paper, and the network embedding problem belong to the more general problem of graph embedding, i.e., constructing a guest graph based on a host graph. As the same class of problems, however, they are applied at different levels and should be analyzed from different perspectives.

Topology reconfiguration lies in the hardware level. From the perspective of manycore processor architects, they reconfigure a virtual topology to isolate various underlying physical topologies so that they can transparently provide OS and programmers a unified interface to ease task dispatching scheduling and application optimization. Network embedding, however, lies in the application level. From the perspective of application programmers, they assume that the underlying system topology is fixed, and then embed their application topology based on the given physical topology to optimize the software performance. If chip architects do not provide a unified (virtual) topology, application programmers should have to handle various embedding problems from their application topology to different chip physical topologies.

It should be noted that, network embedding problems use *dilation* and *congestion* to evaluate the performance of virtual topologies [39]. *Dilation* of a virtual edge e in the guest topology is the length of the corresponding physical path in the host topology. *Congestion* of an edge e in the host topology is the number of virtual edges that include that edge. *Dilation* and *congestion* consider the worst case scenario for the guest topology. However, we use different evaluation metrics in the topology reconfiguration problem in NoC-based manycore systems, i.e., DF and CF. As discussed in Section III, there are a wide range of applications running on the NoC-based manycore systems, it is difficult to evaluate the effect of virtual topologies on various applications at the chip architecture design stage. As a result, we evaluate the performance of virtual topologies themselves. The primary evaluation metric DF, i.e., the average hop count determines the zero-load latency of a virtual topology while the auxiliary metric CF reflects the distribution of traffic load and thus could affect network latency and throughput.

V. Problem Analysis

The objective of TRP in essence is to find a map from virtual locations to physical cores with optimized performance. Considering the configuration shown in Fig. 3, as depicted in Fig. 7, the example virtual topology can be achieved according to the mapping table. For example, virtual location V is mapped to the 2nd physical core. In other words, the 2nd fault-free core is placed in virtual location V in the virtual topology. For the given physical topology in Fig. 7, there are 9! possible virtual topologies with different DF and CF values, because a fault-free core can be placed in any virtual locations.

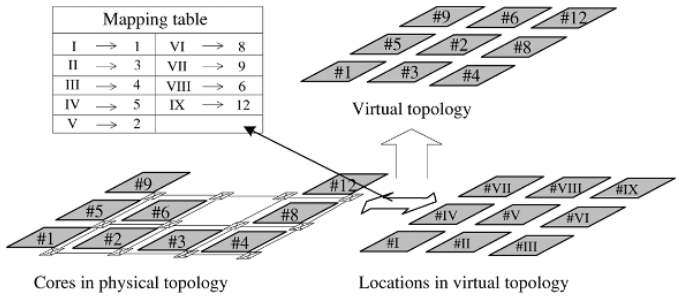


Fig. 7. The essence of TRP is to find a map from virtual locations to physical cores.

The topology reconfiguration problem can be broken into two related subproblems, to minimize DF and to minimize CF, which we call TRP-I and TRP-II, respectively. In this section, we first recast these two problems from an optimization problem to a decision problem, and then show both of them are essentially instances of known NP-complete problems.

A. TRP-I: An Instance of Quadratic Assignment Problem

According to the above analysis, the decision form of TRP-I can be formulated as follows:

[**TRP-I**] Virtual locations are numbered $\{1, 2 \ldots, n\}$, while physical cores are numbered $\{1, 2 \ldots, m\}, n \leq m$. d_{kl} is the distance (number of *hops*) between physical nodes k and l. $d_{kl} = \infty$ if k or l is defective. Is there a one-to-one function $f : \{1, 2 \ldots, n\} \rightarrow \{1, 2 \ldots, m\}$ to construct a virtual topology T, such that: $\mathrm{DF}(T) \leq B$ (bound $B \in Z^+$).

To ease analysis, suppose the reference topology is torus. Each virtual location i has four neighbors in torus. According to (1), the distance factor of i can be expressed as $\mathrm{DF}_i = (1)/(4) \sum_j d_{f(i)f(j)}$, in which j indicates four virtual neighbors of i and $d_{f(i)f(j)}$ represents the physical distance of node i and its virtual neighbors as mentioned above. The above formulation can be similarly applied for mesh topology, except that the coefficients for different nodes can be 1/2, 1/3, or 1/4, as a virtual node in mesh may have 2, 3, or 4 neighbors based on its position.

From the above, according to (2) the distance factor of the virtual topology T is

$$\mathrm{DF}(T) = \frac{1}{4n} \sum_{i=1}^{n} \sum_{j} d_{f(i)f(j)}. \tag{5}$$

We now show that TRP-I is essentially an instance of Quadratic Assignment Problem (QAP), which is a well-known NP-complete problem [40]. QAP can be formulated as follows [41].

[**QAP**] Non-negative integer cost: $c_{ij}, 1 \leq i, j \leq n$; and distance $d_{kl}, 1 \leq k, l \leq m$. Is there a one-to-one function $f : \{1, 2 \ldots, n\} \rightarrow \{1, 2 \ldots, m\}$ such that: $\sum_{i=1}^{n} \sum_{j \neq i}^{n-1} c_{ij} d_{f(i)f(j)} \leq B$.

A QAP instance can be expressed as $\{\langle c_{ij}, d_{kl}, B\rangle, c_{ij}, d_{kl}, B \in Z^+; 1 \le i,j \le n; 1 \le k,l \le m\}$. The famous "backboard wiring" problem [42] is a typical application of QAP, which concerns how to place computer components to minimize the total amount of wiring required to connect them.

Considering a QAP instance $\{\langle c_{ij}, d_{kl}, B\rangle\}$, let i and j be virtual locations $(1 \le i,j \le n)$ in torus, and d_{kl} is the distance between physical nodes k and l as defined in TRP-I. c_{ij} is defined as follows:

$$\begin{cases} c_{ij} = 1/4n, & \text{if } i \text{ and } j \text{ are virtual neighbors} \\ 0, & \text{otherwise.} \end{cases}$$

Then the objective of this QAP becomes

$$\frac{1}{4n}\sum_{i=1}^{n}\sum_{j} d_{f(i)f(j)} \le B \tag{6}$$

in which j are four virtual neighbors of i. According to (5) and (6), it is clear that the objective of the above QAP instance becomes to find a mapping function or in other words a virtual topology (T) with distance factor not exceeding B. As a result, TRP-I is an instance of the quadratic assignment problem.

B. TRP-II: An Instance of Vectorial Quadratic Assignment Problem

Similarly, the decision form of TRP-II can be formulated as follows.

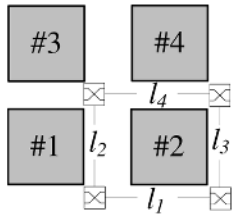
[**TRP-II**] Virtual locations are numbered $\{1,2\ldots,n\}$, while physical cores are numbered $\{1,2\ldots,m\}$, $n \le m$. Is there a one-to-one function $f : \{1,2\ldots,n\} \to \{1,2\ldots,m\}$ to construct a virtual topology T, such that: $\mathrm{CF}(T) \le B$ (bound $B \in Z^+$).

In this subsection, we show that TRP-II is also an instance of quadratic assignment problem, but with a different form. To prove this, we first define a Vectorial Quadratic Assignment Problem (V-QAP) as follows:

[**V-QAP**] Non-negative integer cost: $c_{ij}, 1 \le i,j \le n$; P-dimensional non-negative vector $v_{kl}, 1 \le k,l \le m$, and bound $B_V = (A_1, A_2 \ldots, A_P)$. For two P-dimensional vectors V_1 and V_2, $V_1 \le V_2$ is defined as $|V_1| \le |V_2|$. Is there a one-to-one function $f : \{1,2\ldots,n\} \to \{1,2\ldots,m\}$ such that: $\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij} v_{f(i)f(j)} \le B_V$.

An instance of V-QAP can be expressed as $\{< c_{ij}, P, v_{kl}, B_V >, c_{ij} \in Z^+, v_{kl}$ and B_V are P-dimensional non-negative vectors, $1 \le i,j \le n, 1 \le k,l \le m\}$. It is easy to see that V-QAP is NP-complete because QAP is in fact one-dimensional V-QAP. We now show that TRP-II is an instance of V-QAP. Suppose the reference topology is 2D mesh or torus with L physical links, denoted as $l_1, l_2, l_3 \ldots, l_L$.

Definition 1: Path Vector p_{rs} is a L-dimensional vector $(l_1, l_2, l_3 \ldots, l_L)$. If $l_x (1 \le x \le L)$ is on one of the paths from physical node r to s according to the NoC's routing mechanism (e.g., XY-routing), l_x in p_{rs} is 1, otherwise l_x is 0. A simple example is shown in Fig. 8, in which XY-routing is used. For example, $p_{14} = (1,0,1,0)$ because packets from 1st core to 4th core pass through links l_1 and l_3.



Path vector: p_{rs}=(l_1, l_2, l_3, l_4)	
p_{12}=(1, 0, 0, 0)	p_{31}=(0, 1, 0, 0)
p_{13}=(0, 1, 0, 0)	p_{32}=(0, 0, 1, 1)
p_{14}=(1, 0, 1, 0)	p_{34}=(0, 0, 0, 1)
p_{21}=(1, 0, 0, 0)	p_{41}=(0, 1, 0, 1)
p_{23}=(1, 1, 0, 0)	p_{42}=(0, 0, 1, 0)
p_{24}=(0, 0, 1, 0)	p_{43}=(0, 0, 0, 1)

Fig. 8. Path vector examples.

Definition 2: Congestion Increment Vector v_{rs} is defined as $v_{rs} = p_{rs} - I \times d_{rs}/L$. d_{rs} is the distance between physical node r and s as defined in TRP-I. I is the L-dimensional unit vector.

We now construct a V-QAP instance $\{< c_{ij}, L, v_{rs}, \sqrt{L-1} \times B_V >, 1 \le i,j \le n, 1 \le r,s \le m\}$, in which i and j are virtual locations, and c_{ij} is defined as

$$c_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are virtual neighbors} \\ 0, & \text{otherwise.} \end{cases}$$

According to the definition of V-QAP, we want to find a one-to-one function $f : \{1,2\ldots,n\} \to \{1,2\ldots,m\}$ such that $\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij} v_{f(i)f(j)} \le \sqrt{L-1} \times B_V$.

As c_{ij} is 0 if i and j are not virtual neighbors, the objective then becomes $\sum_{i=1}^{n}\sum_{j} v_{f(i)f(j)} \le \sqrt{L-1} \times B_V$, or in another form

$$\frac{1}{\sqrt{L-1}} \left| \sum_{i=1}^{n}\sum_{j} \left(p_{f(i)f(j)} - I \times \frac{d_{f(i)f(j)}}{L} \right) \right| \le |B_V| \tag{7}$$

in which i and j are virtual neighbors.

Based on the above definitions of path vector and the congestion factor of a link in Section III, it is not difficult to derive: $\sum_{i=1}^{n}\sum_{j} p_{f(i)f(j)} = (\mathrm{CF}_{l_1}, \mathrm{CF}_{l_2} \ldots \mathrm{CF}_{l_L})$ and $(1)/(L)\sum_{i=1}^{n}\sum_{j} d_{f(i)f(j)} = \overline{\mathrm{CF}}$. Then, we can conclude from (7) after substitution: $(1)/(\sqrt{L-1})|(\mathrm{CF}_{l_1}, \mathrm{CF}_{l_2} \ldots \mathrm{CF}_{l_L}) - I \times \overline{\mathrm{CF}}| \le |B_V|$, i.e., $CF \le |B_V|$.

It is clear that the above constructed instance of V-QAP is in fact to find a virtual topology (T) with congestion factor not exceeding $|B_V|$. As a result, we have proved that TRP-II is an instance of V-QAP.

To sum up, in this section we point out that TRP is an instance of the quadratic assignment problem, one of the most complex combinatorial optimization problems. We therefore do not hold much hope for finding an exact polynomial time algorithm for its solution. Efficient and effective heuristics are therefore introduced to solve this problem, as shown in the following section.

VI. Proposed Topology Reconfiguration Algorithm

In this section, an advanced Simulated Annealing (SA) algorithm proposed for QAP is firstly adopted to tackle our TRP. This algorithm, however, is quite time-consuming. We therefore present a fast deterministic greedy algorithm, called Row Rippling and Column Stealing (RRCS). Finally, a gSA algorithm is proposed, which outperforms both SA and RRCS algorithms in terms of computing time and the quality of results.

It should be noted that we mainly focus on the reconfiguration algorithms for 2D mesh/torus topologies. Other topologies (e.g., butterfly or fat tree topology) may require different optimization algorithms.

A. An Adopted Simulated Annealing Algorithm (SA)

Since we have proved that topology reconfiguration problem is an instance of the quadratic assignment problem, we can adopt previous heuristic approaches for QAP to tackle our TRP. One such approach that has yielded promising results is simulated annealing [43]–[46]. We adopt one of the most efficient simulated annealing implementations proposed in [43] for QAP to tackle TRP in this paper.

Various simulated annealing algorithms generally differ with respect to neighborhood search, annealing schedule and termination criterion. The adopted SA algorithm uses (4) as the cost function and random virtual topologies as initial solutions. The neighborhood function employed is the widely used "2-exchange". For example, if the current solution is

$$\begin{bmatrix} (1,0) & \text{Faulty} & \text{unused} \\ (0,0) & (0,1) & (1,1) \end{bmatrix}$$

one of its neighbors by exchanging (1,1) and 'unused' is

$$\begin{bmatrix} (1,0) & \text{Faulty} & (1,1) \\ (0,0) & (0,1) & \text{Unused} \end{bmatrix}.$$

The neighboring solutions are searched thoroughly in a fixed order, not randomly. For the above solution, $5 \times (5-1)/2$ trials are needed to explore all its neighborhood by the sequence (1,0)$\leftrightarrow$'unused', $(1,0) \leftrightarrow (0,0), \ldots$ 'unused' $\leftrightarrow (0,0)$, 'unused' $\leftrightarrow (0,1) \ldots$

The adopted SA algorithm uses the inhomogeneous annealing with oscillation schedules, i.e., temperature is reduced by a very small amount after every trial without any equilibrium test. In addition, temperature is decreased and increased periodically, i.e., reannealing instead of the straightforward annealing, which is the common practice of state-of-the-art simulated annealing algorithms.

The SA algorithm in [43] uses an advanced formula to calculate the initial and final temperatures for each iteration, leaving two tuning control parameters, i.e., the initial (λ_1) and the final (λ_2) temperature factors, which can be used to control the cooling process effectively.

The algorithm terminates when the current iteration number exceeds Q, or in other words after $Qn(n-1)/2$ trials, in which n is the number of fault-free cores.

B. Row Rippling Column Stealing Algorithm (RRCS)

Simulated annealing is a kind of common technique that can be adopted to all combinatorial optimization problems. However, it does not consider any characteristics of the TRP problem, such as reference topology, system architecture, etc. Moreover, SA is quite time-consuming because it has to explore many random solutions before achieving a satisfactory result. As the configuration time has great impact on the chip cost, SA is not acceptable for large scale manycore systems. As a result, we proposed a fast deterministic greedy algorithm, called Row Rippling and Column Stealing (RRCS) [12].

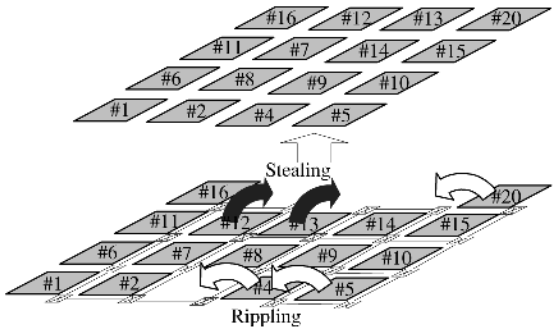


Fig. 9. An example of RRCS algorithm.

RRCS is based on the observation that the performance degradation of a virtual topology is mainly caused by the physical irregularity of the virtual topology compared to the reference topology. Therefore, RRCS algorithm tries to maintain the physical regularity of the virtual topologies in row and in column unit.

To ease illustration, suppose in mesh or torus topology, there are one column of spare cores. If a row contains only one faulty core, i.e., faulty cores are no more than the spare ones in this row, Row Rippling is employed to reconfigure the row, in which a faulty core is replaced by its neighbor and the virtual position of the core used to replace the faulty one is transferred to the next neighboring core. This process continues until the spare one is used to replace the last element in the row. When a row contains more than one faulty cores, i.e., faulty cores are more than the spare ones in this row, the rightmost faulty core is replaced using rippling. The other faulty elements within the row, however, are replaced with the elements immediately beneath them. In other words, we "steal" a fault-free core from another row within the same column. This stolen core should be considered faulty when the row containing it is reconfigured. An example of using RRCS in a "$16+4$" processor with 4×4 mesh reference topology and one column redundancy is depicted in Fig. 9. To configure the uppermost row, which contains 3 faulty cores, we steal the 12th and the 13th fault-free cores for the left two fault cores; while the rightmost one is rippling to the 20th core. Only Row Rippling is used to configure the lowermost row as it contains one faulty core. The achieved virtual topology is shown above the physical topology.

In the above discussion, we provide a column of redundant cores as an example. In practice, the number of redundant cores, i.e., M, for an N-core processor should be carefully determined by the designers in advance (e.g., using the analysis framework in [47]), and may be different from the column size. This however does not affect the working mechanism of the proposed RRCS algorithm as it only needs to compare the number of faulty cores N_f and spare cores on each row. We are able to generate an effective virtual topology as long as the number of faulty cores is less than M. In the worst case, i.e., all available cores in both the same row and the same column are exhausted, we simply choose a nearest core to replace the faulty one.

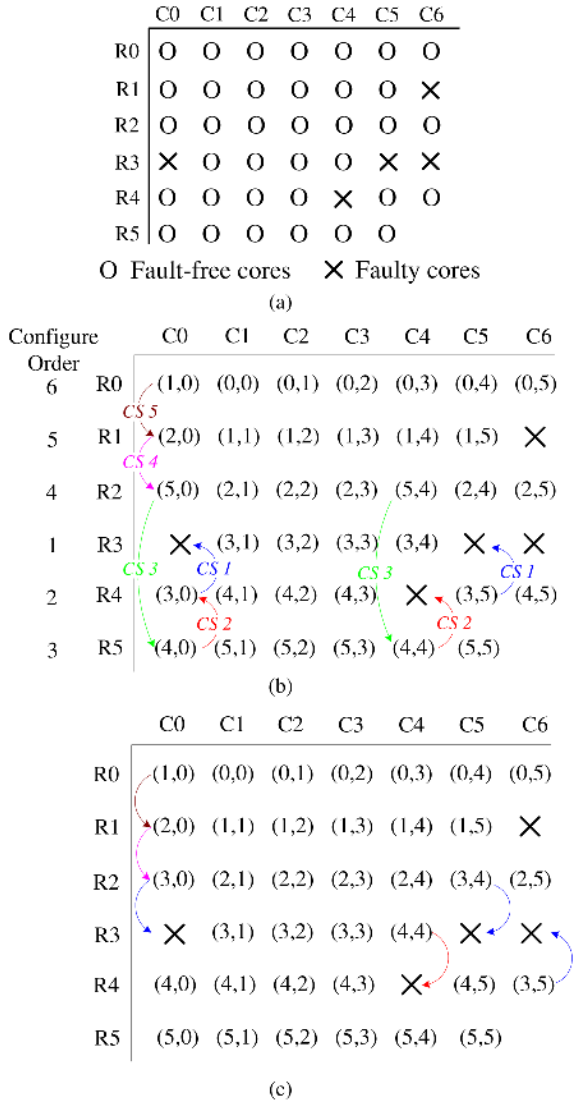


Fig. 10. Comparison between RRCS and gSA. (a) Physical topology. (b) Virtual topology achieved by RRCS $(\mathrm{DF} = 1.660; \mathrm{CF} = 1.428)$. (c) Virtual topology achieved by gSA $(\mathrm{DF} = 1.329; \mathrm{CF} = 0.937)$.

C. RRCS-Guided Simulated Annealing Algorithm (gSA)

RRCS is very fast when compared to SA algorithm, but it does not directly consider DF or CF metrics during the optimization process. Moreover, RRCS may cause serious chain column stealing operations for certain physical topologies and result in undesirable virtual topologies.

For example, consider a physical topology with 6×6 2D mesh reference topology and 5 spare cores located on the right-hand side and 5 faulty cores, as shown in Fig. 10(a). The virtual topology achieved by RRCS is shown in Fig. 10(b), in which the coordinates indicates the virtual locations for the corresponding cores. Reconfiguration begins from row R3, causing two stealing operations, i.e., the two CS1 from row R4. R4 then does not have enough available cores and has to steal another two cores, i.e., CS2 from row R5. The process continues until the last row R0 is configured. Note that CS3 borrows relatively distant cores to configure faulty cores in row R5. These chain column stealing operations will generate an undesirable virtual topology.

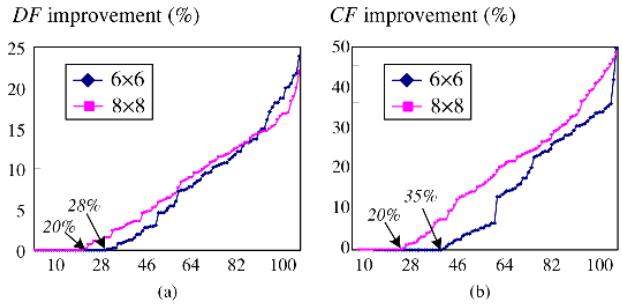


Fig. 11. gSA improvement over RRCS for different network size. (a) DF improvement. (b) CF improvement.

At the same time, RRCS is very efficient, and it can arrange most part of the virtual topology in a good shape. We find that by applying several 2-exchange operations on top of the topologies achieved by RRCS, the quality of the results can be greatly improved. As a result, we propose to combine the algorithms of RRCS and SA together. We use RRCS to quickly generate a good initial solution point, and then apply the adopted SA algorithm on top of it to explore its 2-exchange neighboring solutions. We call this strategy RRCS-guided Simulated Annealing technique (gSA).

We use gSA $(w_{\mathrm{DF}} = 0.9, w_{\mathrm{CF}} = 0.1)$ and RRCS working on 100 random physical topologies in 6×6 2D mesh with 5 spare and 5 randomly distributed faulty cores and 8×8 2D mesh with 8 spare and 8 random faulty cores respectively. The DF and CF improvement of gSA over RRCS are reordered from small to large and are shown in Fig. 11. For the DF metric in 6×6 array, RRCS generates the same results as gSA for the first 28 physical topologies, i.e., no improvement, while for the other 72 cases, gSA has different levels of improvement. When the network size increases to 8×8, gSA achieves greater improvement than in 6×6 for 80% cases. CF metric is similar. We can conclude that RRCS is efficient since for around 20%–35% cases, it generates results as good as gSA. However, for many circumstances due to chain column stealing operations, RRCS has very poor performance, and gSA can improve over RRCS greatly, especially for larger network size.

We then use SA algorithm with different parameters working on the above 100 physical topologies in 6×6 and 8×8 mesh. The initial and final temperature factors λ_1 and λ_2 are tuned and set to be 0.5 and 0.05 respectively. We choose 50 and 100 random solutions, i.e., SA-50 and SA-100 with different iteration numbers, i.e., $Q = 10$ and $Q = 20 \cdot w_{\mathrm{DF}}$ and w_{CF} are set to be 0.9 and 0.1 respectively. The averaged results are shown in Table I. It can be seen that, gSA outperforms SA in all cases with very little computational time. With more random solutions and more iteration numbers, SA improves a little but with great computing time overhead. This is because the quality of random initial solutions used by SA are much worse than RRCS, which is able to focus on a good solution point very fast.

TABLE I
gSA IMPROVEMENT OVER SA FOR DIFFERENT NETWORK SIZE

	6×6 2D mesh with 5 spare and 5 fault cores		
	SA-50 (Q=10)	*SA*-100 (Q=20)	*gSA*
Time(s)	177.4	484.7	2.2
DF	1.538	1.483	1.319
CF	1.396	1.312	0.977
	8×8 2D mesh with 8 spare and 8 fault cores		
	SA-50 (Q=10)	*SA*-100 (Q=20)	*gSA*
Time(s)	484.4	3477.8	8.9
DF	1.782	1.473	1.296
CF	1.615	1.288	0.908

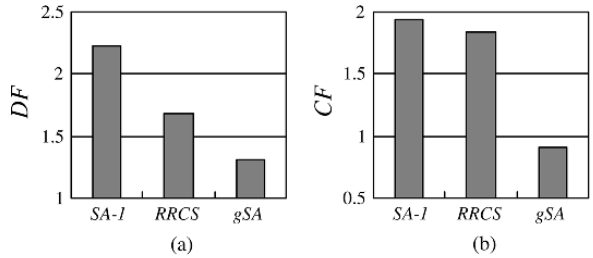


Fig. 12. Comparison between SA-1, RRCS and gSA. (a) DF comparison. (b) CF comparison.

VII. EXPERIMENTAL RESULTS

A. Experimental Setup

We have implemented a manycore NoC simulation platform composed of classic pipelined virtual channel routers and cores which generate synthetic workload. The router pipeline has four stages, i.e., routing computation, virtual-channel allocation, switch allocation and switch traversal, in which each stage takes one clock cycle. Since we want to evaluate the performance of virtual topologies, other parameters should remain unchanged. In our experiments, each physical link has 8 virtual channels, and each virtual channel has 8 flit buffers. Credit-based flow control is used for buffer management. To reveal the performance of topologies themselves, the simple dimension-order routing is used which has the minimum impact on traffic distributions.

As execution-driven workload makes it difficult to isolate bottlenecks in the network design [48] and we concern more about the network performance, we use synthetic workload instead of execution-driven workload. Each core in our manycore NoC simulation platform is actually a traffic generator. As virtual topologies are constructed based on the spatial locality of communication, we adopt the neighboring traffic pattern in our experiments, in which a core only exchanges information with its neighbors. It is important to point out that the traffic patterns are applied to virtual topologies, not to physical topologies. That is, 1-hop communication between virtual neighbors may involve multiple physical hops.

Virtual topologies generated by reconfiguration algorithms are in XML format to be read by the simulation platform. Each core will then be assigned a name "$c\_vtx\_vty\_phx\_phx$", in which (vtx, vty) and (phx, phy) are its virtual and physical coordinates. Each time a core sends a packet, it reads its virtual location, looks up the mapping table stored in the simulator to find the physical locations of its virtual neighbors and then encapsulates in the packets as the destination address.

B. Experiment I

In this experiment, we show how predictive of DF and CF metrics to real performance measurements. DF is the average hop count between virtual neighbors and thus should reflect the average delay and throughput of the network. While CF indicates traffic distribution across all the physical channels. We use SA-1 (1 random initial solution), RRCS and gSA ($w_{\mathrm{DF}} = 0.9, w_{\mathrm{CF}} = 0.1$) to work on 100 different physical topologies in 8×8 2D mesh with 8 spare cores and 8 randomly distributed faulty cores on-chip. We use SA-1 to keep the computational time comparable to gSA. We choose the physical topology on which gSA achieves the greatest improvement over SA-1 and RRCS in this experiment. The obtained DF and CF values are shown in Fig. 12.

Next, we import virtual topologies generated by these three algorithms into our manycore NoC platform to get the simulation performance measurements, i.e., average delay, throughput and average occupied time of all channels as shown in Fig. 13.

Average delay is the time required for a packet to traverse the network from source to destination. It can be observed from Fig. 13(a), the latency of virtual topologies achieved by SA-1, RRCS and gSA are almost the same under light traffic load. When the network saturates, it is clear that the delay of gSA is better than RRCS, and RRCS is better than SA-1. Network throughput is the packets delivering rate for a particular traffic pattern. Fig. 13(b) shows the throughput of saturation of the three algorithms. It is clear that the throughput of gSA is higher than RRCS, while RRCS is higher than SA-1. Compared with Fig. 12(a), we show the effectiveness for DF as performance metrics.

Fig. 13(c) shows the percentage of occupied time of all physical channels. More occupied time implies that more traffic passing through that channel. We reorder these values from small to large for easy comparison. It can be observed that the curve for gSA has the smallest slope, which means the differences between all channels are small, i.e., the traffic is more evenly distributed. RRCS is more steep than gSA, and SA-1 is more steep than RRCS. Compared with Fig. 12(b), we show that the CF metric reflects real performance measurement.

From the above we can conclude that, gSA has better performance than RRCS and SA-1, not only in terms of DF and CF metrics but also in real performance measurements, i.e., latency, throughput and traffic distribution. In addition, the effectiveness of DF and CF as evaluation metrics is proved with this experiment.

C. Experiment II

In this experiment, we evaluate the effectiveness of the proposed gSA algorithm with the scale of network size. We use the 8×8 2D mesh topology with 8 spare cores and 8 randomly distributed faulty cores. We choose another larger configuration with 10×10 2D mesh reference topology, 12 spare cores and 12 random faulty cores for proportional scaling. We work on 100 random physical topologies in 8×8 and 10×10 respectively. The average improvement of gSA over RRCS for DF metric is 6.828% in 8×8 while 9.737% in 10×10 configurations. Regarding the CF metric, the improvement is 18.935% in 8×8

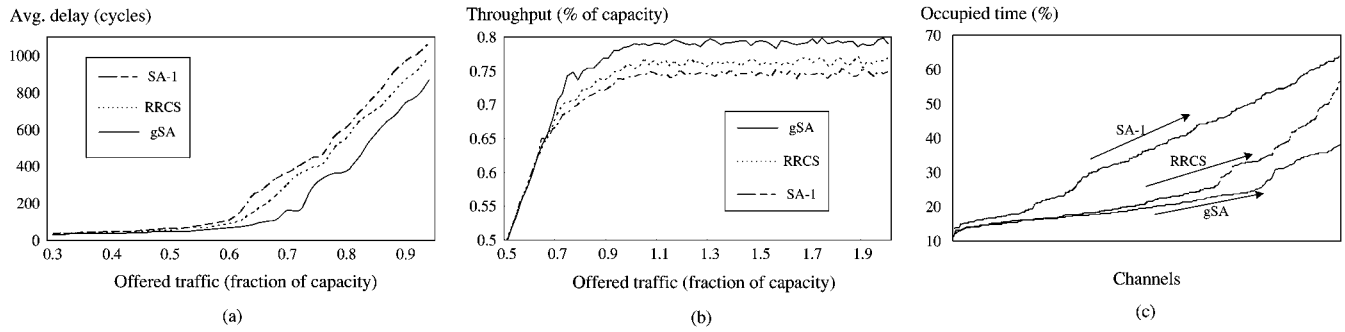


Fig. 13. Simulation measurements comparison between SA-1, RRCS and gSA. (a) Average delay. (b) Throughput. (c) Traffic distribution.

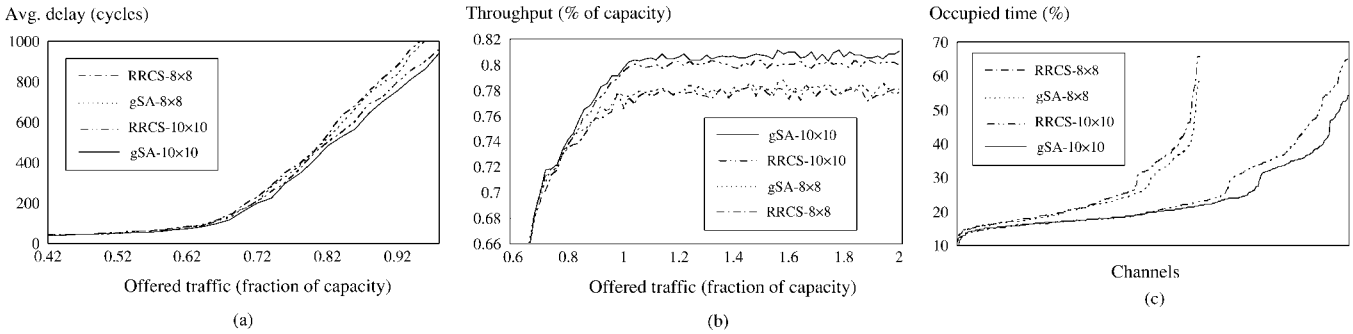


Fig. 14. Comparison between RRCS and gSA for different network size. (a) Average delay. (b) Throughput. (c) Traffic distribution.

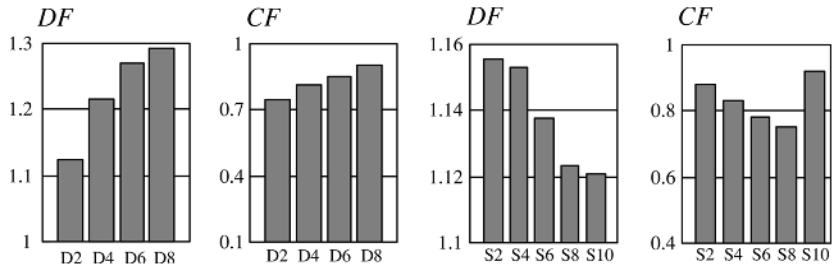


Fig. 15. The impact of different number of faulty cores and spare cores on gSA algorithm.

and 20.983% in 10×10 respectively. That means when network becomes larger, gSA achieves much better improvement over RRCS.

The average delay, throughput and traffic distribution are shown in Fig. 14. It is clear that gSA improves over RRCS for both network sizes. For smaller network size, i.e., 8×8, the averaged delay, throughput and traffic distribution of virtual topologies achieved by RRCS are much closer to that of gSA. For larger network size, i.e., 10×10, gSA achieves much better improvement in all measurements. Thus we can conclude that, firstly, when network size scales, gSA achieves better improvement; secondly, we further validate the effectiveness of DF and CF because the level of improvement for these two metrics and real performance measurements are similar.

D. Experiment III

In this experiment, we evaluate the impact of different number of faulty cores and spare cores on gSA algorithm.

Firstly, we use 8×8 2D mesh with one column spare cores. We vary the number of faulty cores from 2 to 8 (i.e., D2, D4, D6 and D8). Faulty cores are randomly distributed, leading to various physical topologies. Results are averaged and shown in the first two figures in Fig. 15.

It is clear that when the number of defective cores increases, the performance of virtual topologies achieved by gSA slightly becomes worse in terms of both DF and CF. This is expected because the increase of faulty cores limits the solution space of the proposed algorithm.

Next, we assume there are always 2 randomly distributed faulty cores in 8×8 2D mesh and we vary the number of spare cores from 2 to 10 (i.e., S2, S4, S6, S8 and S10). As expected, the increase of spare cores also increases the solution space of the gSA algorithm, and both DF and CF slightly becomes better. However, when the number of spare cores is increased from 8 to 10, we find that DF almost remains the same while CF becomes much worse as in Fig. 15. This is because there are many cores and channels left unused on-chip, traffic distribution becomes much uneven. Therefore, we can conclude employing more-than-necessary number of spare cores does not facilitate to boost the NoC-based manycore systems' performance much after reconfiguration.

VIII. Conclusion and Future Work

Effective defect tolerance techniques are essential to improve the yield of homogeneous manycore processors. In this paper, we propose to employ core-level redundancy with AMAD scheme

to address this issue. As defective cores change the topology of the target design, programmers may face various different topologies when optimizing their parallel programs. This is a big burden and may also cause confusion in marketing. We propose to address the above problem by providing a unified topology that is isomorphic with the target reference topology regardless of the various possible underlying physical topologies. We borrow the concept of virtual topology from network embedding problem and we propose two metrics to evaluate the performance of different virtual topologies. An effective heuristic, namely Row Rippling Column Stealing-guided Simulated Annealing algorithm is then presented to solve the topology reconfiguration problem. The proposed algorithm is evaluated on various topologies in a NoC-based manycore simulation platform. Experimental results not only show the effectiveness of the proposed gSA algorithm, but also show the effectiveness of the two evaluation metrics used in our algorithms, i.e., DF and CF.

In our future work, we plan to investigate the topology reconfiguration problems for topologies other than mesh and torus (e.g., butterfly topology).

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] D. Geer, "Chip makers turn to multicore processors," *IEEE Computer*, vol. 38, no. 5, pp. 11–13, May 2005.

[2] S. Borkar, "Thousand core chips—A technology perspective," in *Proc. ACM/IEEE Design Automation Conf.*, Jun. 2007, pp. 746–749.

[3] A. Agarwal and M. Levy, "The kill rule for multicore," in *Proc. ACM/IEEE Design Automation Conf.*, Jun. 2007, pp. 750–753.

[4] Intel From a Few Cores to Many: A Tera-Scale Computing Research Overview [Online]. Available: http://techresearch.intel.com/articles/Tera-Scale/1421.htm

[5] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. ACM/IEEE Design Automation Conf.*, Jun. 2001, pp. 18–22.

[6] I. Koren and Z. Koren, "Defect tolerance in VLSI circuits: Techniques and yield analysis," *Proc. IEEE*, vol. 86, no. 9, pp. 1819–1838, Sep. 1998.

[7] I. Koren, "Should yield be a design objective?," in *Proc. Int. Symp. Quality of Electronic Design*, Mar. 2000, pp. 115–120.

[8] Ed. Sperling, "Turn Down the Heat Please," [Online]. Available: http://www.edn.com/article/CA6350202.html Mar. 2007

[9] I. Koren and D. K. Pradhan, "Yield and performance enhancement through redundancy in VLSI and WSI multiprocessor systems," in *Proc. IEEE*, May 1986, vol. 74, no. 5, pp. 699–711.

[10] P. Shivakumar, S. W. Keckler, C. R. Moore, and D. Burger, "Exploiting microarchitectural redundancy for defect tolerance," in *Proc. IEEE Int. Computer Design Conf.*, Oct. 2003, pp. 481–488.

[11] E. Schuchman and T. N. Vijaykumar, "Rescue: A microarchitecture for testability and defect tolerance," in *Proc. IEEE/ACM Int. Symp. Computer Architecture*, Jun. 2005, pp. 160–171.

[12] L. Zhang, Y. Han, H. Li, and X. Li, "A fault tolerance mechanism in chip many-core processors," in *J. Tsinghua Science and Technology*, Jul. 2007, vol. 12, no. S1, pp. 169–174.

[13] I. Parulkar, T. Ziaja, R. Pendurkar, A. D'Souza, and A. Majumdar, "A scalable, low cost design-for-test architecture for UltraSPARC™ chip multi-processors," in *Proc. IEEE Int. Test Conf.*, Oct. 2002, pp. 726–735.

[14] P. J. Tan, T. Le, K.-H. Ng, P. Mantri, and J. Westfall, "Testing of UltraSPARC T1 microprocessor and its challenges," in *Proc. IEEE Int. Test Conf.*, Oct. 2006, Paper 16.1.

[15] S. Makar, T. Altinis, N. Patkar, and J. Wu, "Testing of Vega2, a chip multiprocessor with spare processors," in *Proc. IEEE Int. Test Conf.*, Oct. 2007, Paper 9.1.

[16] "Microsoft," Application Software Considerations for NUMA-Based Systems [Online]. Available: http://www.microsoft.com/whdc/system/platform/server/datacenter/numa_isv.mspx

[17] A. Gupta, A. Tucker, and S. Urushibara, "The impact of operating system scheduling policies and synchronization methods of performance of parallel applications," in *Proc. ACM SIGMETRICS Measurement and Modeling of Computer Systems Conf.*, May 1991, pp. 120–132.

[18] S. L. Scott and G. M. Thorson, "The Cray T3E Network: Adaptive routing in a high performance 3D torus," in *Proc. Hot Interconnects IV*, Aug. 1996, pp. 147–156.

[19] S.-Y. Kuo and W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays," in *Proc. ACM/IEEE Design Automation Conf.*, June 1986, pp. 385–390.

[20] D. K. Bhavsar, "An algorithm for row-column self repair of RAMs and its implementation in the Alpha 21264," in *Proc. IEEE Int. Test Conf.*, Sep. 1999, pp. 311–318.

[21] T. Kawagoe *et al.*, "A built-in self-repair analyzer (CRESTA) for embedded DRAMs," in *Proc. IEEE Int. Test Conf.*, Oct. 2000, pp. 567–574.

[22] X. Du, S. M. Reddy, W.-T. Cheng, J. Rayhawk, and N. Mukherjee, "At-speed built-in self-repair analyzer for embedded word-oriented memories," in *Proc. IEEE Int. VLSI Design Conf.*, Jan. 2004, pp. 895–900.

[23] C.-T. Huang, C.-F. Wu, J.-F. Li, and C.-W. Wu, "Built-in redundancy analysis for memory yield improvement," *IEEE Trans. Reliability*, vol. 52, no. 4, pp. 386–399, Dec. 2003.

[24] S.-K. Lu, Y.-C. Tsai, C.-H. Hsu, K.-H. Wang, and C.-W. Wu, "Efficient built-in redundancy analysis for embedded memories with 2-D redundancy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 1, pp. 34–42, Jan. 2006.

[25] A. Jain, B. Mandava, J. Rajski, and N. C. Rumin, "A fault-tolerant array processor designed for testability and self-reconfiguration," *IEEE J. Solid-State Circuits*, vol. 26, no. 5, pp. 778–788, May 1991.

[26] S.-Y. Kung, S.-N. Jean, and C.-W. Chang, "Fault-tolerant array processors using single-track switches," *IEEE Trans. Computers*, vol. 38, no. 4, pp. 501–514, Apr. 1989.

[27] J. H. Kim and P. K. Rhee, "The rule-based approach to reconfiguration of 2-D processor arrays," *IEEE Trans. Computers*, vol. 42, no. 11, pp. 1403–1408, Nov. 1993.

[28] T. A. Varvarigou, V. P. Roychowdhury, and T. Kailath, "Reconfiguring processor arrays using multiple-track models: The 3-track-1-spare-approach," *IEEE Trans. Computers*, vol. 42, no. 11, pp. 1281–1293, Nov. 1993.

[29] Y.-Y. Chen, S. J. Upadhyaya, and C.-H. Cheng, "A comprehensive reconfiguration scheme for fault-tolerant VLSI/WSI array processors," *IEEE Trans. Computers*, vol. 46, no. 12, pp. 1363–1371, Dec. 1997.

[30] C. P. Low, "An efficient reconfiguration algorithm for degradable VLSI/WSI arrays," *IEEE Trans. Computers*, vol. 49, no. 6, pp. 553–559, Jun. 2000.

[31] W. Jigang and S. Thambipillai, "An improved reconfiguration algorithm for degradable VLSI/WSI arrays," *J. Systems Architecture: The EUROMICRO J.*, vol. 49, no. 1–2, pp. 23–31, Jul. 2003.

[32] M. Fukushi, Y. Fukushima, and S. Horiguchi, "A genetic approach for the reconfiguration of degradable processor arrays," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Syst.*, Oct. 2005, pp. 63–71.

[33] D. Doval and D. O'Mahony, "Overlay networks: A scalable alternative for P2P," *IEEE Internet Computing*, vol. 7, no. 4, pp. 79–82, Jul./Aug. 2003.

[34] B. Cong, L. Cong, and S. Q. Zheng, "Lower bounds of network embedding dilations," in *Proc. Midwest Symp. Circuits Syst.*, Aug. 1993, vol. 1, pp. 558–561.

[35] F. Liu and L. Xu, "The topological properties and network embedding of RP(k)," in *Proc. Int. Conf. Parallel and Distributed Computing, Applications and Technologies*, Dec. 2005, pp. 21–25.

[36] MPI: A Message-Passing Interface Standard, Version 2.0, [Online]. Available: http://www.mpi-forum.org/

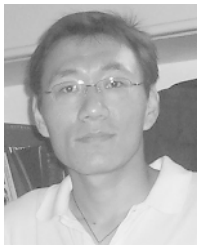
[37] S. Moh, C. Yu, H. Y. Youn, B. Lee, and D. Han, "Mapping strategies for switch-based cluster systems of irregular topology," in *Proc. Int. Conf. Parallel and Distributed Syst.*, 2001, pp. 733–740.

[38] N. Bauch and E. Maehle, "Reconfiguration in octagonal mesh-based multicomputer systems with distributed checkpointing," in *Proc. IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, Jun. 1994, pp. 169–180.

[39] S.-Y. Kim and J. Hur, "An approach for torus embedding," in *Proc. Int. Workshops on Parallel Processing*, 1999, pp. 301–306.

[40] S. Sahni and T. Gonzalez, "P-complete approximation problems," *J. Assoc. Comput. Machinery*, vol. 23, no. 3, pp. 555–565, Jul. I976.

[41] M. R. Garey and D. S. Johnson, *Computer and Intractability—A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.
[42] L. Steinberg, "The backboard wiring problem: A placement algorithm," *SIAM Rev.*, vol. 3, no. 1, pp. 37–50, Jan. 1961.
[43] A. Misevicius, "A modified simulated annealing algorithm for the quadratic assignment problem," *Informatica*, vol. 14, pp. 497–514, 2003.
[44] R. E. Burkard and F. Rendl, "A thermodynamically motivated simulation procedure for combinatorial optimization problems," *Eur. J. Oper. Res.*, vol. 17, pp. 169–174, 1984.
[45] D. T. Connolly, "An improved annealing scheme for the QAP," *Eur. J. Oper. Res.*, vol. 46, pp. 93–100, 1990.
[46] A. Bolte and U. W. Thonemann, "Optimizing simulated annealing schedules with genetic programming," *Eur. J. Oper. Res.*, vol. 92, pp. 402–416, 1996.
[47] S.-J. Pan and K.-T. Cheng, "A framework for reliability analysis of system fault tolerance and component test quality," in *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE) Conf.*, Apr. 2007, pp. 1–6.
[48] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA: Morgan Kaufmann, 2003.
[49] L. Benini and G. D. Micheli, *Networks on Chips, Technology and Tools*. San Mateo, CA: Morgan Kaufmann, 2006.
[50] L.-T. Wang, C.-W. Wu, and X. Q. Wen, *VLSI Test Principles and Architectures*. San Mateo, CA: Morgan Kaufmann, 2006.
[51] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. San Mateo, CA: Morgan Kaufmann, 2004.
[52] W. Stallings, *Operating Systems: Internals and Design Principles*, 4th ed. Englewood Cliffs, NJ: Prentice-Hall.



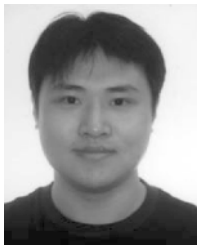
Lei Zhang received the B.Eng. degree in computer science from the University of Electronic Science and Technology of China (UESTC), Sichuan, China, in 2003, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2008.

He is currently an Assistant Professor at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include network-on-chip, design for reliability and multicore/manycore processors.

Yinhe Han (M'06) received the B.Eng. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2001, and the M.Eng. and Ph.D. degrees in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2003 and 2006, respectively.

He is currently an Associate Professor at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include VLSI/nano-electronic/MEMs design and test, reliability design and advanced digital measurement.

Dr. Han was a recipient of the Test Technology Technical Council Best Paper Award at the Asian Test Symposium 2003. He is also a member of ACM/IEICE.



Qiang Xu (S'03–M'05) received the Ph.D. degree in electrical and computer engineering from McMaster University, Canada, in 2005, and since then has been an Assistant Professor of Computer Science and Engineering at The Chinese University of Hong Kong.

Dr. Xu leads the CUhk REliable computing laboratory (CURE Lab.). His research interests range from test and debug of system-on-a-chip integrated circuits to fault tolerance and reliable computing. He has published more than 40 technical papers in these areas. He received the Best Paper Award in 2004 IEEE/ACM Design, Automation and Test in Europe Conference (DATE). He is a member of the ACM SIGDA, the IEEE, and the IEEE Computer Society. He has served as a technical program committee member for a number of conferences on VLSI design and testing.

Xiaowei Li (SM'04) received the B.Eng. and M.Eng. degrees in computer science from Hefei University of Technology, China, in 1985 and 1988, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 1991.

From 1993 to 2000, he was an Associate Professor in the Department of Computer Science, Peking University, China. During 1997 and 1998, he was a Visiting Research Fellow in the Department of Electrical and Electronic Engineering, University of Hong Kong. During 1999 and 2000, he was a Visiting Professor in the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. He joined the Institute of Computing Technology, Chinese Academy of Sciences, as a Professor in 2000. His research interests include VLSI testing, design verification, and dependable computing.

Dr. Li serves as a member of the Editorial Board of the *Journal of Computer Science and Technology* and the *Journal of Low Power Electronics*, an Associate Editor-In-Chief of the *Journal of Computer-Aided Design and Computer Graphics* (in Chinese). He was a Technical Program Chair of IEEE Asian Test Symposium (ATS) in 2003, and Workshop of RTL and High Level Testing (WRTLT) in 2001. He was a General Chair of ATS in 2007, WRTLT in 2003. He serves on the Technical Program Committee of several IEEE and ACM conferences, including VTS, DATE, ASP-DAC and PRDC. He also serves as Asia-Pacific Regional TTTC Vice-Chair.

Huawei Li (M'00) received the B.S. degree in computer science from Xiangtan University, Hunan, China, in 1996, and the M.S. and Ph.D. degrees in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 1999 and 2001, respectively.

She is currently a Professor at the Institute of Computing Technology, Chinese Academy of Sciences. Her research interests include VLSI/SOC design verification, test generation, delay test and design for reliability.